Design and Implementation of a Wi-Fi Portal System

Tobias Asplund 36039
Supervisors: Annamari Soini, Dragos Truscan
Faculty of Science and Engineering
Åbo Akademi University 2021

Abstract

The goal of the thesis is to develop a proof-of-concept of a Wi-Fi portal service to evaluate the feasibility and scope for creating a fully-fledged Wi-Fi portal solution. A Wi-Fi portal is the webpage that users are required to interact with to gain access to a public Wi-Fi network. The focus of the work was to build the framework for a Wi-Fi portal system to be deployed on Cisco Meraki access points and create the user interfaces for editing and managing portals. The proof-of-concept has support for creating and customizing portals in a WYSIWYG (what you see is what you get) editor, with a live preview of the created portal. The portal page has support for authenticating through email, SMS and Facebook authentication. Additional third-party services can be implemented in a similar manner. Out of a selection of alternatives for which technologies to use for the portal, Golang with the Golang template packages was deemed to be the best solution for serving the portal. Using Golang to generate the HTML on the server allows for moving most of the logic away from users devices and moving possible points of failure to a controlled environment. The proof-of-concept fulfilled the requirements set up at the start of the project. A complete product can be developed using the findings in the thesis and the proof-of-concept as a base.

Keywords: Wi-Fi, portal, splash page, public Wi-Fi

Acknowledgements

I would like to thank my supervisors Annamari Soini and Dragos Truscan for their guidance in the writing of this thesis, especially for providing feedback more frequently and thoroughly that anyone could expect. I would also like to thank my friends who gave valuable feedback to provide another perspective when I got stuck on my own writings. Last but not least I would like to thank my colleagues for having patience with my sporadic days off for writing this thesis.

Tobias Asplund
Turku, May 22, 2021

Contents

Glossary

Access point A device that allows wireless to connect to a network.

API Application Programming Interface.

Captive portal synonym for Wi-Fi portal, see Wi-Fi portal.

CRM Customer Relationship Management.

CSS Cascading Style Sheets.

Data cap Limitation on data transferred over a mobile connection.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

GET A HTTP request used to request data from a specified resource.

HTTP HyperText Transfer Protocol.

HTTPS HyperText Transfer Protocol Secure.

JSON JavaScript Object Notation.

MAC Media Access Control.

MVP Minimum Viable Product.

POC Proof-of-concept.

POST A HTTP request used to send data to a server to create/update a resource.

PUT A HTTP request used to send data to a server to create/update a resource.

RFP Request For Proposal.

SDK Software development kit.

SSID Service Set Identifier.

UI User Interface.

URL Uniform Resource Locator.

WAI Walkbase Analytics Intelligence.

Wi-Fi portal A Wi-Fi portal is the webpage that users are required to interact with to gain access to a public Wi-Fi network.

Wi-Fi splash page synonym for Wi-Fi portal, see Wi-Fi portal.

WLAN Wireless Local Area Network.

1. Introduction

Wi-Fi portals represent the de facto way to authenticate users in a public Wi-Fi environment. A Wi-Fi portal, sometimes called a Wi-Fi splash page, is the page that greets users when they try to sign on to a public Wi-Fi network that requires authentication. The authentication can range from just agreeing to terms and conditions to signing in with a social media account or buying access to the Wi-Fi network.

Even with the required initial investment in infrastructure, public Wi-Fi is commonplace in today's retail locations. Providing public WiFi in retail locations can be an arguable investment, where both security concerns and the costs have to be weighed against the benefits provided by offering it. The customer demand for public Wi-Fi is due to the high rate of smartphone adoption. The GSMA 2019 report [1] states that smartphone adoption 2019 among the population in North America was 80 percent and 72 percent in Europe, with a prediction of 90 and 82 percent respectively for 2025. Even though the use of personal mobile internet connections such as 4G are common, there is no guarantee that they will work at reasonable speeds in highly congested areas or inside structures such as shopping centers where cellular reception can be limited. The connection speeds and pricing of personal mobile connections also vary greatly between countries and regions [2]. Many providers enforce data caps, which limits the data a user can transfer over the connection, effectively steering users to use public Wi-Fi where it is offered. In a survey done by Purple [3], 72 percent of recipients answered that they use Wi-Fi in public spaces and 63.36 percent answered that they are more likely to spend more in a venue that offers public Wi-Fi.

The benefit of getting customers to spend more time in a location by offering public Wi-Fi is already a good starting point, but there are missed opportunities if the only thing the public Wi-Fi does is provide Internet access. Adding a Wi-Fi

portal to the public Wi-Fi solution can help with both gathering useful metrics for key performance indicators and adding new channels to reach customers. With a Wi-Fi portal, useful information and marketing material can be placed on the users' screens as they join the network and contact information and user metadata can be gathered during the sign-in process.

For venue proprietors to benefit the most from providing a public Wi-Fi network, they need to be able to authenticate users and gather statistics about usage and their customers. The authentication can range from asking the user to provide an email or phone number, register an account on the venue owner's online service to using a third-party authentication provider such as Facebook or Google. The data gathered during authentication can provide an insight into customer segments and behavior. The portal page that users interact with when signing in to the Wi-Fi network is also a valuable communication channel that can be utilized for information or product recommendations.

The company Walkbase has started a project to develop an in-house solution for providing a Wi-Fi portal system that can integrate with a multitude of Wi-Fi access points from different vendors. The goal of this thesis is to develop a proof-of-concept (POC) Wi-Fi portal service which can be incorporated into the Walkbase Audience Intelligence product, enabling a direct-to-consumer marketing channel for venue proprietors such as retailers and quick service restaurants. The proof-of-concept includes an editor for creating and editing portal pages, a system to manage and assign them to different locations and Wi-Fi access points, and the necessary software infrastructure to render and serve the portals. A complete product will support Wi-Fi access points from multiple manufacturers, however, the proof-of-concept will focus on Cisco Meraki access points. Gathering of data and providing analytics are not included in the proof-of-concept, but will be implemented in future versions.

Figure 1.1: A Wi-Fi portal is a way to authenticate and communicate with users that connect to a public Wi-Fi network. Users interact with the portal in the form of a webpage on their device.

There are two different users referred to in this thesis. The end user is a user that interacts with a Wi-Fi portal page while connecting to a Wi-Fi network. A typical end user is an individual visiting a store or other public space. The other user is an individual who creates and manages Wi-Fi portals with the solutions created in this thesis. This person is typically working in a support role for a company that provides the portal system, or in a role at a retailer that involves editing the portal content.

1.1 Background

1.1.1 Walkbase Analytics

Walkbase analytics [4] is an in-store analytics solution that provides venue proprietors with the tools to understand how their customers behave in the physical space. It makes it possible to make data-driven decisions and adapt for sales performance and customer loyalty. Walkbase analytics aggregates data from Wi-Fi, Bluetooth beacons and other in-store sensor technologies. The Walkbase dashboard is the main tool with which venue proprietors interact with Walkbase Analytics. The dashboard provides a summary of locations and key performance indicators in one place as well as fine-grained analytics. A promotional image for the Walkbase Analytics dashboard can be seen in Figure 1.2.

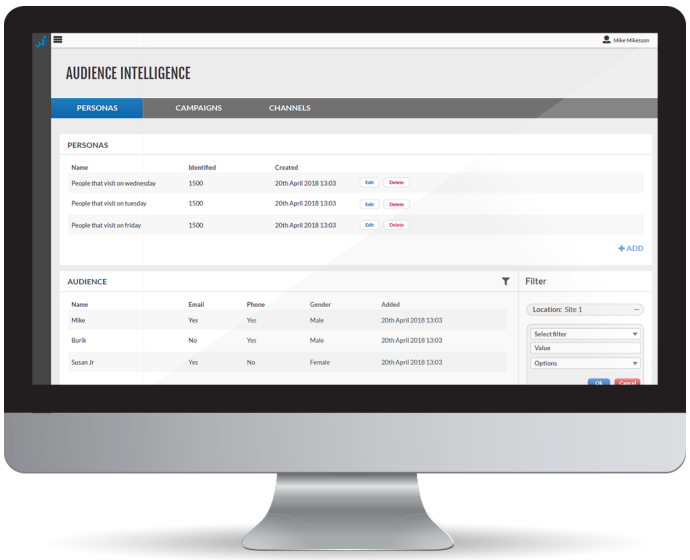

Figure 1.2: Promotional image for the Walkbase Analytics dashboard.

Walkbase Audience Intelligence (WAI) adds a CRM (customer relationship management) and marketing layer to the Walkbase Analytics Product. Walkbase Audience Intelligence helps venue operators understand their customers by providing deeper knowledge about visiting patterns by customer segments. It also adds the

possibility to communicate with customers through a variety of channels. Personas can be created that represent a specific customer segment. These personas can be used as target groups in campaigns to send marketing messages to through email, SMS, push notifications, and third-party marketing solutions.

The Wi-Fi portal solution is planned as an addition to the Walkbase Audience Intelligence package. It will allow retailers to keep track of how their Wi-Fi is used and gain a better insight into their customer base and customer behavior. It can also provide new channels for retailers to communicate with their customers, such as providing information about bonus programs or acquiring customer contact details through the portal page. Figure 1.3 displays a promotional image for WAI in the Walkbase Analytics dashboard.

The existing Walkbase products already utilize Wi-Fi access points for in-store analytics, which makes a portal system a natural extension of the current product offerings. In contrast to standalone portal systems, an in-house system can easily be integrated with the rest of Walkbase analytics products. The data gathered through the portal system can be used together with the currently available analytics data to provide an even deeper insight into how customers behave in retail locations. The portal system is also a potential marketing channel for WAI campaigns. The WAI campaigns feature is a tool that allows targeting specific personas with marketing messages through available channels.

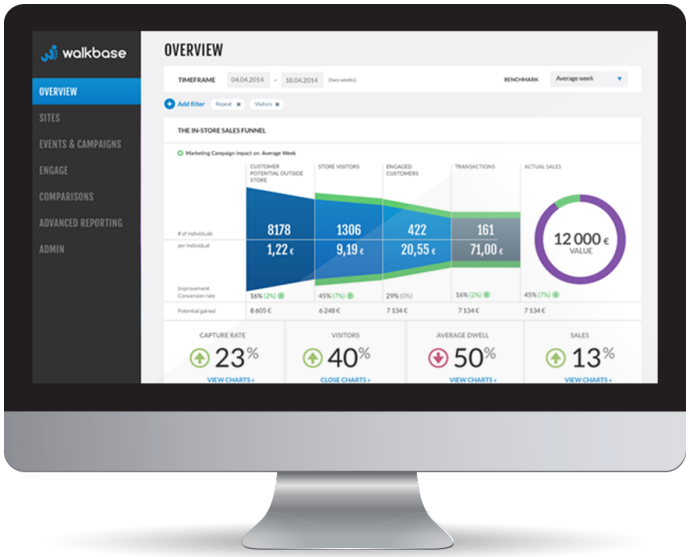

Figure 1.3: Promotional image for Walkbase Audience Intelligence in the dashboard.

1.1.2 Wi-Fi Portal

A Wi-Fi portal, commonly referred to as a *captive portal* or *splash page*, is a web page that users are required to interact with to gain access to a public Wi-Fi network. The interaction required to gain access to the network ranges from simply viewing the page and agreeing to an end-user license agreement to paying for access to the network. When the required interaction is completed, the user is allowed to access the network. In addition to authentication, the portal page can hold any other information desired by the operator of the Wi-Fi portal, such as information about the venue or advertisements.

Most devices today automatically check for Wi-Fi portals when they connect to a Wi-Fi network and can redirect the user to a Wi-Fi portal browser page for authentication. They do this by performing a DNS (domain name system) check to a known domain after connecting to a network. Apple iPhone devices use *captive.apple.com* [5] and Android devices use *connectivitycheck.gstatic.com* [6][7] and a few other domains. If the client device receives an HTTP (HyperText Transfer Protocol) [8] redirect response from the DNS check, it assumes the connection is behind a Wi-Fi portal. An HTTP redirect is a special response to a request that in-

dicates that the client should continue to another location. The device then opens a browser window with the Wi-Fi portal for the user to interact with. If the device does not check for Wi-Fi portals or fails to open the portal automatically, it will still be shown when a web page is requested with a browser on the device. This works since any HTTP requests made while not authenticated will be redirected to the portal page by the access point. Figure 1.4 provides a high level visualization of the captive portal check flow. HTTPS (HyperText Transfer Protocol Secure) requests are still an issue, since they can not be redirected. Internet draft RFC 7710 [9] contains a solution to this by adding Wi-Fi portal options to HTTP requests to more reliably inform devices that they are required to authenticate through a portal.

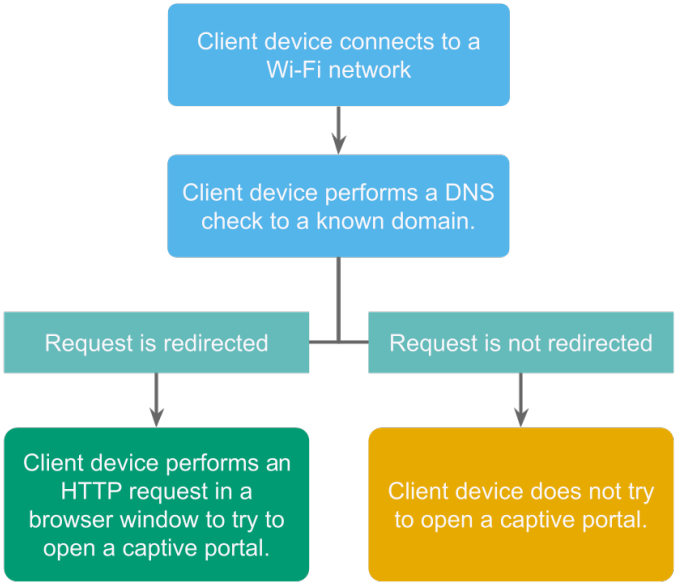

Figure 1.4: Client device captive portal check.

1.1.3 Wi-Fi Access Points

Wi-Fi access points are a central piece of a Wi-Fi system, and there are a variety of manufacturers. A wireless network access point is a physical hardware device that allows wireless devices to connect to a network. It exposes a Wi-Fi network associated with an SSID (service set identifier) that the client devices can wire-

lessly connect to. The SSID is seen by users as the network name. The access point can handle DHCP (Dynamic Host Configuration Protocol) and routing on the WLAN (wireless local area network) and provide access for the connected devices to other networks or the Internet. Contrary to a network router that creates a local network and communicates with other networks, an access point extends an existing network and gives connected devices access to that network. Access points can be individually managed for personal use or centrally managed for enterprise solutions. For the proof-of-concept of the Wi-Fi portal solution, this thesis will focus on Cisco Meraki [10] access points and how to configure them. Figure 1.5 displays a picture of such a device. The long-term goal is to support most of the other common brands on the market.

Figure 1.5: The Cisco Meraki MR32 Cloud-Managed - wireless access point used for the proof-of-concept version of the portal system.

1.1.4 Cisco Meraki Access Points

The Cisco Meraki access points are managed through a cloud-based Meraki dashboard. The part of their network management that is relevant for this thesis is the ability to configure Wi-Fi portals. The access points already have a portal page feature built in, but also have the possibility to configure a self-hosted portal [11]. The built-in portal feature allows for customizing a theme, a message, and a logo. The self-hosted portal is what we are looking at in this thesis, to provide venue proprietors with full control over what is shown and how data related to the portals is used.

To configure a custom portal in the Cisco Meraki dashboard the system needs a URL (uniform resource locator) that points to where the custom portal service is running. The URL also needs to be whitelisted in the walled garden in the Meraki dashboard. A walled garden allows for restricting access to a specific set of IP addresses or hostnames [12]. Any additional sources that are used in the portal page, such as third-party authentication, need to be whitelisted as well. If these addresses are not whitelisted the client device will not be able to access them, since this happens before the device is given unrestricted access to the internet. The whitelisting can be done either with IP addresses or domain names.

When an unauthorized end-user device connects to a Cisco Meraki access point configured to use a custom portal page, any HTTP requests are intercepted and forwarded to the Meraki cloud service. The service then forwards the request to a preconfigured URL that hosts our portal service. Figure 1.6 provides a visualization of how the access point handles a connection request.

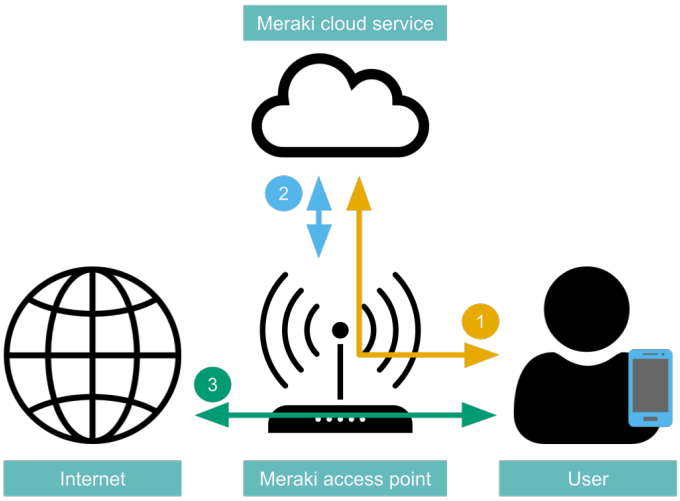

Figure 1.6: 1) A client device makes a request which is intercepted by the access point and forwarded to the Meraki cloud service. 2) When the required interactions are completed the Meraki cloud service notifies the access point that the client device is allowed to access the internet. 3) The client device can now access the internet.

If the portal service implementation needs any additional information from the end-user device, that information can be added to the preconfigured URL as additional parameters. It is up to the portal page to manage authentication and interaction with the user. When the user has completed the required interactions, which might be signing in through a third party or simply clicking a button, the Meraki cloud service notifies the access point that the user is authenticated and the access point gives the user internet access. This process is described in depth in chapter 2.

1.2 Goal and Validation Criteria

The goal for this project is to extend the existing Walkbase platform to include the necessary tools and infrastructure for the creation and management of Wi-Fi portals. The work includes both formulating the necessary functionality for a minimum viable product (MVP) and creating the proof-of-concept solution where a user can create and manage the WiFi portal content for Meraki access points. The requirements for both the proof-of-concept solution and the MVP are listed in section 1.3. The focus of the work is to build the framework for the Wi-Fi portal and create the user interfaces. This work was done in collaboration with other Walkbase team members, with for example the necessary APIs (Application Programming Interfaces) being implemented by other developers based on the requirements presented in this thesis.

We need to create a user interface for venue proprietors to control the content and configuration of their Wi-Fi portals. A single proprietor might have multiple locations that need customized content in their Wi-Fi portals. The configurations need to be saved somewhere and retrieved when a portal is served, and for that we need a storage solution for storing the portal configurations. To serve a portal page there needs to be a hosted backend service that generates the portal HTML [13] according to the saved configurations and serves it to the end user.

Based on the proof-of-concept solution, Walkbase will be able to evaluate the feasibility and scope of work for creating a fully-fledged solution where the Wi-Fi portal content and interactions can be in their entirety managed through Walkbase Audience Intelligence portal. The scope of this thesis will, however, be limited to the proof-of-concept solution. The MVP solution requirements are listed here to provide a complete picture of all the required components and functionality.

1.3 Project Requirements

1.3.1 Portal Format

The portal, or the configuration for a portal, needs to be created, stored, retrieved, and edited. The configuration will be stored in the existing Walkbase infrastructure and accessed through an API created specifically for the portal service. The storage and API are created by other parts of the team at Walkbase and are not covered in depth in this thesis. While the portal editor is not concerned by the format in which the configuration is stored in the backend, the format received from the API should be a JSON (JavaScript Object Notation) object containing all the necessary information needed to render the portal page. The actual format of the configuration is developed together with the team at Walkbase.

To be able to reach as large an audience as possible the webpage served to the client device, hereby referred to as portal page, needs to be lightweight and load reasonably fast even on old and outdated devices. To make these design decisions we need to find out which browsers and devices are used and what limitations they have. In section 2.1 we will look at browser and device statistics with a focus on statistics gathered from the US market, since that is where the first customers will be. We will also look at European and world-wide statistics where relevant.

1.3.2 Portal Editor

The portal editor is the tool used by venue proprietors to create and customize the look and feel of their Wi-Fi portals. The editor will be incorporated into the existing Walkbase dashboard, hence the same frameworks that are used in the dashboard will be used to create the portal editor. A requirement for the editor is that it is a WYSIWYG (what you see is what you get) editor. This means that there needs to be a live preview of the portal that updates when the configuration changes, so that it matches what the real portal looks like.

1.3.3 Authentication

The portal will allow users to authenticate through third-party authentication such as Facebook, Instagram, Twitter, Linkedin, and Google, or email and SMS. For the proof-of-concept version of the the portal only Facebook authentication will be implemented. It should be possible to implement the other third-party authentications in a similar way once one of them is implemented.

1.3.4 General Requirements

The technologies chosen should work well with the current infrastructure used for Walkbase products. Incorporating new technologies or programming languages would require them to be substantially better than the currently available alternatives to incorporate into the current technology stack used at Walkbase. The solution is aimed to work on the types of mobile devices and operating systems with a large enough user base to capture the majority of users. The devices, browsers, and operating systems to support should be based on usage statistics; there is no need to specifically target support for a device if it is no longer commonly used by end users.

1.3.5 Portal Content Editing and Creation Requirements

These requirements were provided at the start of the project and other requirements might arise during development. The requirements are given from a user interface standpoint. Although not directly mentioned in the requirements, the listed requirements imply that the features they describe should be supported in the APIs as well. All of the requirements listed in Table 1.1 and Table 1.2 are relevant for the MVP and a majority, but not all, are relevant for the POC. They are split up between the management of portals and the editing of a portal.

In the portal management tool, a user of the portal system, such as a venue operator, needs to be able to create a new portal and define a descriptive name for the

portal. Allowing custom images to be uploaded and possibly edited is a part of the MVP, but for the POC it is enough to provide a predefined set of images to choose from in the editor. After having created a portal, it should be possible to open it again and edit it. It should also be possible to delete or archive portals that are no longer needed.

The editor should be a WYSIWYG editor with at least the feature set described here. The portal should be visually customizable by allowing for either creating or selecting a theme that includes colors, fonts, and other visual aspects. For the POC it is enough to be able to customize the visuals, while predefined themes to choose from are only required for the MVP. Sign-in options need to be customizable as well, with the possibility of selecting from a list of available sign-in options. There needs to be a gallery view for images from which they can be added to the portal page in a desired location. There should be at least one area in the page where the user can freely add content such as text. A requirement for the MVP but not the POC is to simulate different types of screens in the editor preview, to give a preview of what the portal will look like on a phone, tablet, and computer screen.

Table 1.1: Requirements related to the management of portals

Management requirements		
1	Create new portal	A user should be able to create a new portal through the portal management interface.
2	Name a portal	A user should be able to add and change the name of a created portal.
3	Upload images	A user should be able to upload images in the portal management interface and use uploaded images in created portals.
4	Open existing portal for editing	A user should be able view and edit already created portals.
5	Delete a portal	A user should be able to delete or archive created portals.

Table 1.2: Requirements related to the portal editor

Portal editor requirements		
6	A content area with WYSIWYG type editor	The editor should be a WYSIWYG type editor. The user should see a real or simulated preview of the portal.
7	Selecting a dedicated area for sign-up options	The user should be able to select where on the page the sign in options should be displayed.
8	Place images	The user should be able to place images on the portal page and specify size and position.
9	Choose or create a visual theme	The user should be able to create or choose a visual theme to style the portal in a desired way.
10	Simulate different device types	The editor should be able to simulate different screen types such as mobile, tablet, and desktop in the preview.

1.3.6 Portal Assignment Requirements

The portal assignment system will not be implemented for the POC, but as it is necessary for the MVP, the requirements listed in Table 1.3 to provide a complete picture of the planned system, even though it is outside the scope of the POC created in this thesis. For the POC the portal assignment system will be substituted by manually assigning portal URLs (Uniform Resource Locator) to access points in the Meraki cloud service.

The functionality of the portal assignment system consists mainly of assigning created portals to Wi-Fi access points in physical locations. It should be possible to assign a portal to a specific venue or a whole region or group of venues, instead of having to assign it to every access point in a venue. In Walkbase systems, a venue refers to a physical location with Walkbase solutions installed. It should be possible to enable and disable assigned portals. It might not be desirable to always have the portal accessible by customers, hence, it should be possible to

set active hours for a portal. The MAC (Media Access Control) addresses of the access points are already used and available in Walkbase systems, therefore, using the MAC address to identify which venue or region an access point belongs to and which portal it should use can be figured out based on the MAC address for the MVP instead of a hardcoded URL as in the POC.

Table 1.3: Requirements related to assigning portals to Wi-Fi access points in physical locations

Assignment requirements	
11 Connect a portal to venues	If a portal is assigned to a venue, it should be assigned to all access points in the venue.
12 Connect a portal to regions	Is a portal is assigned to a region, it should be assigned to all access points in venues in that region.
13 Enable/Disable a portal	It should be possible to enable and disable a portal access by venue. If a portal is disabled the public Wi-Fi will be unavailable until it is enabled.
14 Choose active times for a portal	It should be possible to enable and disable a portal according to a schedule.

2. Design Considerations

This chapter discusses the different alternatives for the implementation. Possible solutions range from using modern frontend frameworks; such as Angular [14] or simply pure JavaScript, to managing the rendering of the application on the server. The portal editor will be written in Angular, since it will be a part of the Walkbase dashboard, which is made in Angular. With the portal editor, venue proprietors must be able to define a configuration for their portals. The configuration consists of authentication options, styles, colors, texts, and possibly images used in the portal. For the editor to be a WYSIWYG editor, a preview of the portal in the editor needs to update in real time when settings are changed. How the portal adjusts to the changes depends on which technologies were chosen to create it. If the portal is rendered on the client device, it can be included and served with the dashboard, whereas if it is rendered on the server, the editor can either run the real portal in a frame in the editor or use a mock portal that just mimics the look and feel of the real one.

2.1 Browser Usage Statistics

Statistics for browser versions and devices are gathered from Statcounter Global Stats [15]. Statcounter is a web analytics service which has tracking code running on over two million sites globally. For this project, the statistics are used to determine if there are any common limitations in devices that might access the Wi-Fi portal and how that impacts the decisions made for the project. Figure 2.1, Figure 2.2, and Figure 2.3 display the collected data for browser, Android, and iOS versions.

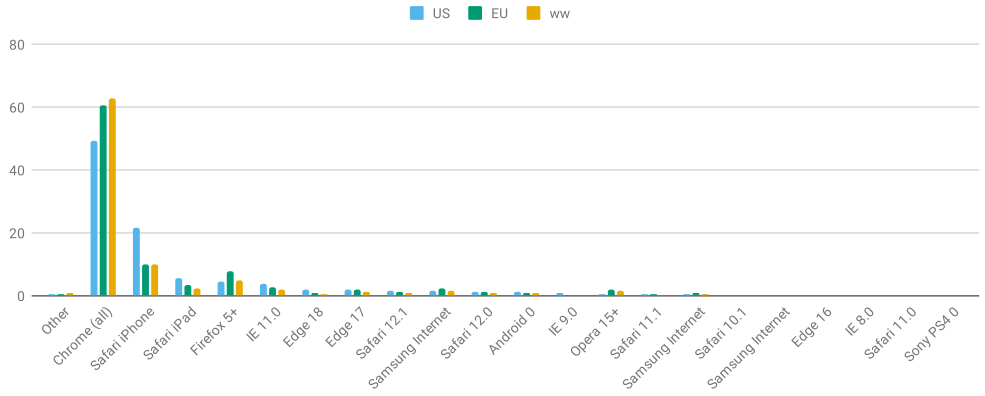

Figure 2.1: The browser usage for the US, the EU, and worldwide.

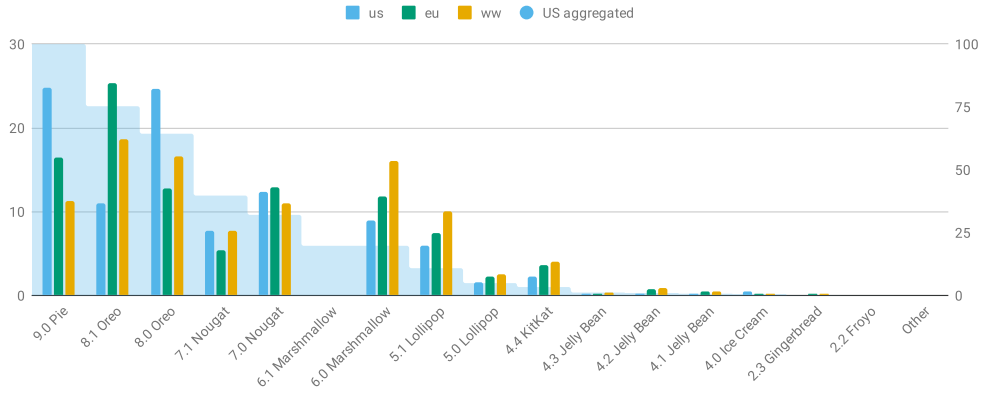

Figure 2.2: Android version statistics in the US, the EU, and worldwide.

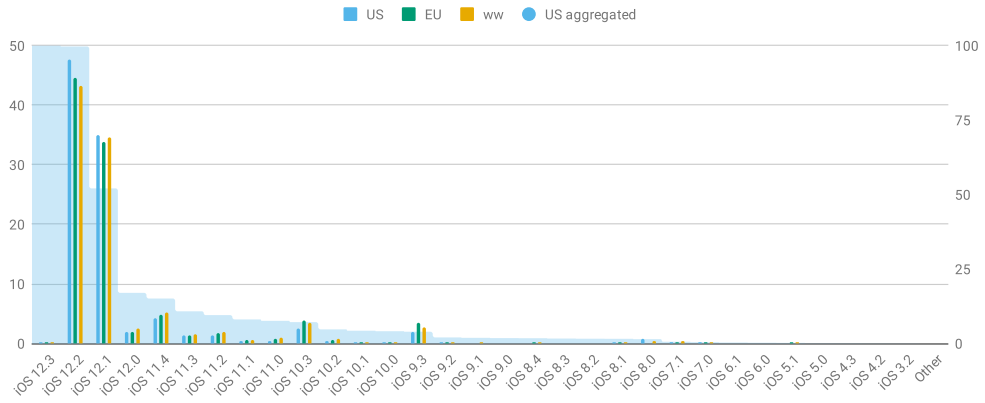

Figure 2.3: iOS version statistics in the US, the EU, and worldwide

From Figure 2.1, we can see that the Chrome browser has an overwhelming majority, with Safari for iPhone and iPad in second place. The few top used browsers already cover enough of the market not to need to go deeper into the limitations of the browsers with a smaller market share. If there is any uncertainty during development of how well a specific CSS (Cascading Style Sheets), JavaScript or HTML feature is supported on older devices, it should be checked and possibly avoided if there are better supported features that can be used instead. Figure 2.2 and Figure 2.3 cover the distribution of Android and iOS versions. The oldest Android version with any notable representation is 4.4 Kitkat, which is not supported by some of the alternatives discussed in the next section. For iOS, an overwhelming majority of devices are running one of the three latest versions of the operating system and should not present any legacy-related support issues for the portal development.

2.2 Evaluation of Technological Alternatives

The technologies listed in this section are the alternatives for creating the portal itself. They are gathered based on existing knowledge within Walkbase and are also limited by what can be integrated into the current infrastructure at the company. Some noteworthy technologies that have been left out are Node.js, Vue.js and React. Node.js is a JavaScript runtime environment that executes JavaScript on the server, but which does not fit into the current infrastructure. Vue.js and React are both quite similar to Angular, which is already used within the company, hence Angular is preferred if such a solution is selected. Some of the listed technologies include both backend and frontend solutions. Any alternative that only includes one of these would have to be combined with a compatible match.

2.2.1 Angular Ccomponent

Creating the portal in Angular is an obvious alternative to consider, since some of the existing Walkbase products are written in Angular. It would require no new

technologies to be learned and maintained by Walkbase developers. Since the dashboard that the portal editor will be a part of is written in Angular, the WYSIWYG part of the editor for portals could be easily implemented by including the actual portal Angular component in the editor, passing it the configuration object to be rendered live in the dashboard. Angular is also one of the popular frontend frameworks [16] and allows for more advanced features to be implemented in the future, if the need should arise. The drawback of using Angular is the limited official browser version support [17] and the fact that a fully-fledged frontend framework such as Angular can be heavy on the client device compared to less complex alternatives. Only the latest versions of major browsers are officially supported. On Android devices, the official support goes as far back as KitKat (4.4) which was released in October 2013. The Android versions older than KitKat only stand for 1.16 percent, as can be seen in Figure 8. Support on iOS is worse where only the two latest major versions are supported, which leaves out about seven percent of iOS users, as can be seen in Figure 9. The term "supported" in Angular assures that new versions of Angular are tested in those browsers and operating systems, although they might work adequately in older versions as well, but there is no guarantee of that.

2.2.2 HTML With Pure JavaScript

Pure JavaScript [18], often referred to as vanilla JavaScript, is JavaScript without additional frameworks. Refraining from modern frameworks and their advanced features would allow JavaScript to be run in almost any browser, even as far back as Netscape. Some users choose to disable JavaScript in their modern browsers to prevent all types of tracking enabled by JavaScript. Going down this road leaves out all nice-to-have and quality-of-life features provided by modern frameworks. This alternative could be combined with a backend solution that would handle the generation of the portal page, leaving out the need for advanced features of large frameworks.

2.2.3 HTML Without JavaScript

Supporting exactly all browsers is only possible to do without any JavaScript at all. This requires the use of server-side rendering to apply the configuration for the portal. Leaving out JavaScript also leaves out third party authentication options that do not have a JavaScript-free alternative for their authentication page, although the JavaScript-powered authentication options only need to be left out for those users who have disabled JavaScript in their browsers. This solution would leave all the interactivity of the portal to be handled by the backend.

2.2.4 JQuery 1.x

JQuery 1.x [19] could be used as a compromise between pure JavaScript and a modern framework. JQuery 1.x is the first major version of JQuery, which was developed for browsers that today are considered old. It is still used today to make web applications with support for legacy browsers. JQuery is a JavaScript framework that simplifies HTML document traversal and manipulation for the developer, without forcing or even encouraging the use of specific programming patterns. JQuery is a good option if the frontend requires more advanced features that are tedious to create with only JavaScript without supporting frameworks.

2.2.5 Hugo

Hugo [20] is a static site generator written in Golang [21]. It allows rendering the HTML webpage on the server instead of using JavaScript frameworks that put a larger load on the client device. After a quick prototype, it became clear that Hugo was not a good choice for the requirements of this project. Hugo works by creating static HTML files every time there is a change in the content. Since the portals need to be edited and saved, to then be fetched from a database would require keeping track of any changes in the configurations and somehow triggering a rerender of the HTML files.

2.2.6 Django

The Walkbase administration tool is written in Django [22], which means no new backend technologies would be required if the portal should be written in Django. Django is a web framework written in python. Django is a large robust framework with a big feature set. With features relating to most parts of web development it claims to be scalable from small to large projects. The portal page itself is a very minimalistic project compared to large data-driven websites. Since the services handling and storing the data for the portal system are separate from the service providing the user interface, the large set of features provided by Django might not be necessary.

2.2.7 Golang

Golang [21] is already used in the Walkbase backend and it can be incorporated into an existing Walkbase project without the need for supporting new technologies. The Golang template packages [23][24] provide data-driven templates for generating text and HTML output. The Golang *text/template* package implements data-driven templates for generating textual outputs based on a data structure. The Golang *html/template* package wraps the *text/template* package for protection against HTML injection. Many Walkbase services are already written in Golang and Golang has strong acceptance within the company.

2.2.8 Selected Alternative

With server-side rendering it is possible to keep the state and logic on the server-side instead of having to do it in JavaScript on the client device. Not having to run JavaScript on the client device also removes any compatibility issues with devices that do not support or have limited support for JavaScript. If the logic that would require JavaScript to run on the client device can be executed on the server instead, more possible points of failure on unknown devices are moved to a controlled environment. Considering that, only the alternatives with server-

side rendering solutions remain for consideration. The remaining contenders are Django and Golang with the template packages. Both of those alternatives are good options for the project, moving as much of the logic as possible from the frontend to the backend. There is no clear stronger alternative between those two. With Django being a full stack framework while the portal service is a relatively simple service, we decided to use Golang with the Golang template packages.

With the majority of the logic moved to the backend there is no need for a complex framework on the frontend. The frontend will simply run the HTML generated by the Golang template packages and if any need for interactivity arises it can be solved with vanilla JavaScript. Some third party authentications might require their own JavaScript bundles to be included, which are discussed more in section 2.3.

2.3 Authentication

For the proof-of-concept of the portal; only Facebook authentication will be supported, together with basic email and SMS forms. Email and SMS will be available as options in the user interface but will not send any actual emails or text messages.

An issue for email- and SMS-based authentication is that a user might not always have access to these services to receive a verification code. It is possible to whitelist email traffic to allow a user to access emails before authenticating through the Wi-Fi portal, although this is not always desired. For the proof-of-concept, both SMS and email will just require a basic sanity check of the format instead of sending an actual verification code.

To notify the access point that the chosen authentication process is completed, the portal redirects the user to a URL provided by Meraki that tells Meraki that the user should get access to the internet. This URL is added to the request that is forwarded to the portal URL when an unauthenticated user tries to make a request through the access point.

For third-party authentication, the access points need to be configured to let requests through to domain names used by the third-party authentication. This is done by whitelisting the relevant authentication URLs in the Meraki dashboard walled garden.

Facebook provides two ways of doing third-party authentication in browsers. The recommended way is to include the Facebook JavaScript SDK (software development kit) in the page and manage authentication through the APIs in the SDK. The other way to provide Facebook authentication is to manually build a login flow based on redirects [25].

When implementing Facebook authentication in the recommended way using the SDK, I noticed that Firefox and possibly other modern browsers block the Facebook SDK as a part of content blocking. The content blocking is intended to protect users from excessive tracking and harmful content. Therefore, I had to implement a manual flow instead.

The Facebook SDK provides a user interface for users to sign in with Facebook and handles the communication with Facebook. When setting up the manual flow no external JavaScript needs to be included; instead, the authentication is managed by sending the user to a Facebook sign-in page (Figure 2.4) hosted by Facebook. When the user has signed in with a Facebook account, the sign-in page sends the user back to the initial page with the required information.

To start the manual login the user is redirected to a Facebook authentication address that takes three parameters, *client_id*, *redirect_uri*, and *state*. The ID is the app identifier that can be found in Facebook's developer dashboard. This requires a developer to sign up for a Facebook developer account [26]. The redirect URI is the address where the user will be redirected after a successful or failed sign-in. This page will have to handle the success or failure to sign in with an appropriate response. The state parameter is there for the portal page to keep track of the state, such as any user or session identifiers necessary to identify the user through the redirects. There are also optional parameters, *response_type* and *scope*. *Re-*

sponse_type is used to determine if the data included when redirected back occurs in URL parameters or fragments. *Scope* is a list of permissions requested by the portal page, the default permissions provide access to the public Facebook profile and friends list. To check if a user has canceled the sign-in, check for error parameters which are *error*, *error_reason*, and *error_description*.

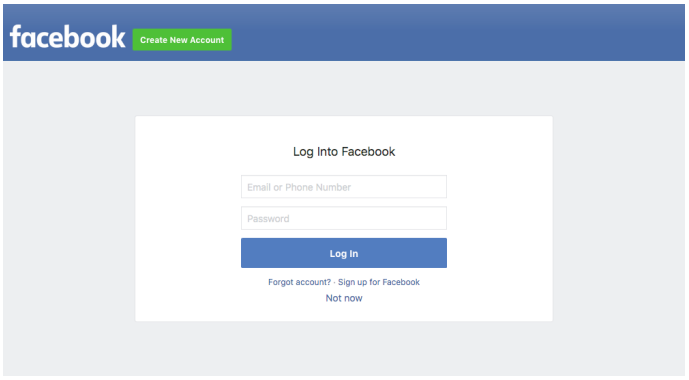

Figure 2.4: The Facebook login page to which a user is directed to authenticate with Facebook.

2.4 WYSIWYG Editor

A WYSIWYG editor (what you see is what you get) is an editor that provides a preview of how the edited content will be displayed in its published environment. The preview can be an exact match or a very close resemblance. There are three possible ways to implement the WYSIWYG editor for the portal page, depending on the technologies used to create the portal. The first alternative is to load in the actual portal in a frame in the preview area. To be able to show the real portal page in a frame the configuration of the portal needs to be passed to the service that serves the portal. This requires that the portal backend either supports applying a configuration per request, or that a mode for editing is supported where the edited configuration is saved in the same way as when saving a portal and then loaded in the preview frame. In that case, the portal would not need to be aware that it is showing a preview and would just behave the same as it would in a production environment.

The second alternative is to only use the same or similar HTML template and CSS for both the preview and the portal. The preview would then only mimic the look and feel of the portal. This alternative avoids the added complexity of the other solutions; instead, it requires maintaining duplicate versions of the visual parts of the portal.

The last alternative is to have a portal that supports both server side and client-side rendering. This could be done by using server side JavaScript to apply the configuration for the portal while also being able to run the same code client-side in the dashboard. This alternative is heavily restricted by the choice of technology for creating the portal.

2.5 Portal Communication Flow

This section describes the full portal communication flow, from the end-user device through the Wi-Fi access point to the portal service and other services involved. It starts when the end-user device connects to a public Wi-Fi network and sends out an HTTP request to a known address. The Wi-Fi access point to which the device is connected forwards the request to the Meraki cloud service which, in turn, forwards it to the preconfigured URL pointing to the portal service. The portal service generates a portal according to the portal configuration and serves it as an HTML web page to the client device. The user can now see the portal and is able to select a way to authenticate. If the user selects a third-party authentication method, such as Facebook, the user is sent to that service for authentication. If the user selects a non-third-party authentication method, such as email or SMS, the relevant form is fetched from the portal service.

When the user has provided the relevant information for authentication, the information is sent to the portal service for validation. The user is then redirected to the *base_grant_url*, which is a URL defined by Meraki cloud service to which a user will be directed to be granted internet access. When the user accesses that URL, the Meraki cloud service notifies the Wi-Fi access point to allow the client to

access the internet. The user is then redirected by the Meraki cloud service to the *continue_url* which is the success page of the portal. Figure 2.5 describes the full portal communication flow.

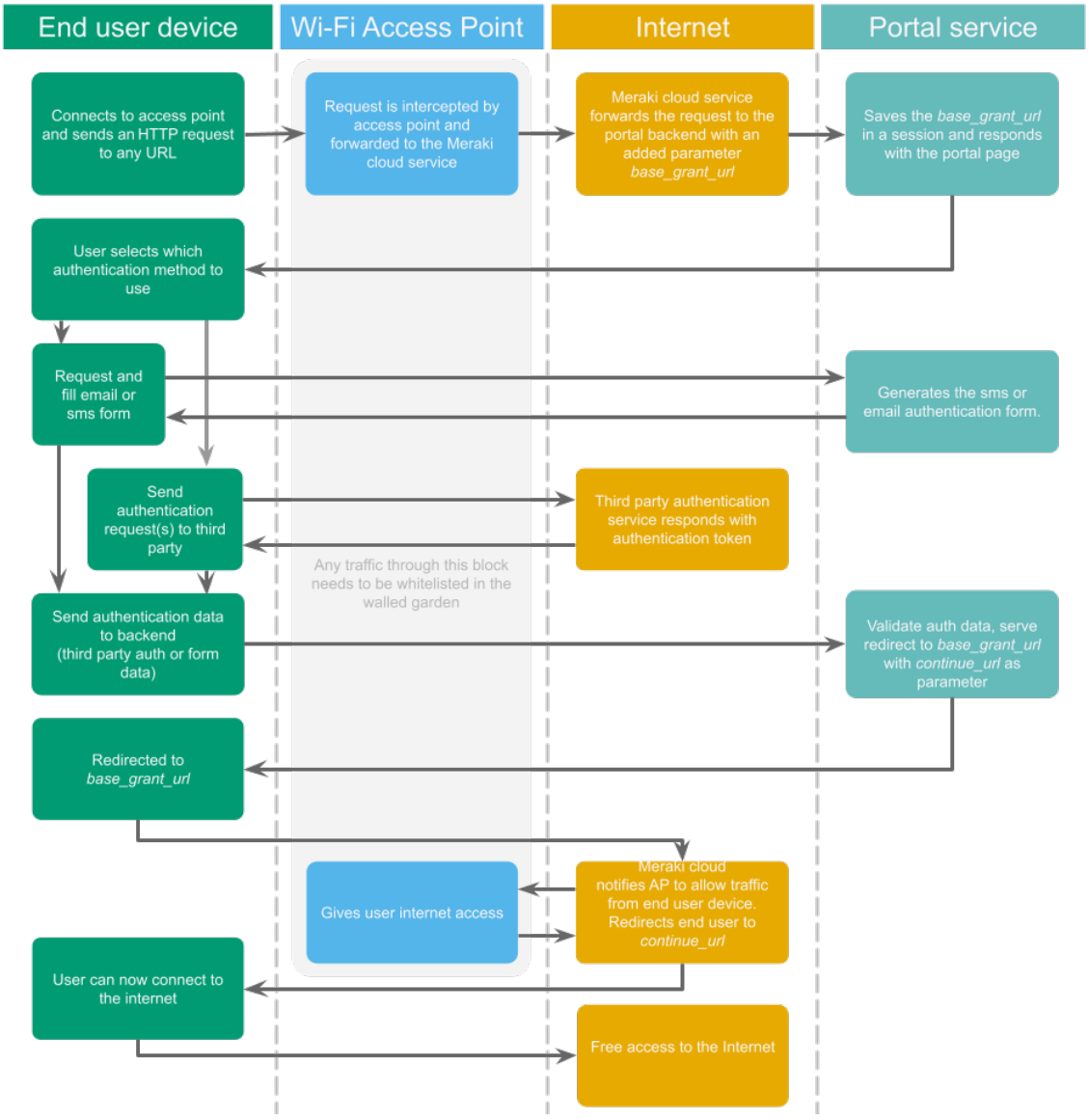

Figure 2.5: The portal communication flow, starting from the first connection and ending with internet access for the user.

3. Architecture and Implementation

The implementation consists of parts developed in this thesis and parts that are only touched on in the thesis and developed by other parts of the team at Walkbase. The portal editor, which will be incorporated into the Walkbase Dashboard, and the portal service that serves the portal to the client device are implemented in this chapter. The APIs and the underlying storage are developed by other members of the team at Walkbase and incorporated into existing backend solutions. Figure 3.1 describes the portal editor, service, and APIs to provide a high-level representation of the implemented solution.

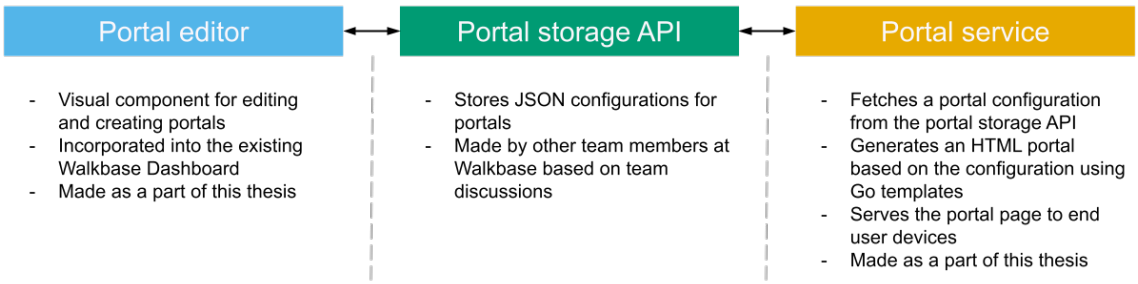

Figure 3.1: High-level overview of the project.

3.1 Portal

3.1.1 Configuration

The portal, or the configuration of a portal, needs to be stored in a format that is both easy to edit in the portal editor and easy to render. Backwards compatibility also needs to be considered when extending the portal configuration in the future. The configuration used when editing and rendering the portal is a JSON object with all the necessary information. In this section, we will discuss the different attributes in the configuration.

There are four attributes used for identification in the configuration. These are *id*, *uuid*, *revision_id*, and *current*. The *ID* is used for internal identification within the portal system. The *uuid* stands for universal unique identifier and is a 128 bit number used for external identification such as when specifying which portal an access point should request. The reason for using a *uuid* is that a larger identifier is less error prone than a simple short number. It is difficult to generate a valid *uuid* by mistake or by guessing, compared to a normal identifier that usually starts from zero and is incremented by one for each new entry. The *revision_id* is used to keep track of revisions of the portal configuration. A new *revision_id* is generated every time the portal is edited and saved. This allows for going back to a previous version of the configuration, if any unwanted changes are made. The *current* attribute simply indicates if the configuration is the current one, since newer versions are kept even if the configuration is reverted to a previous one.

The attribute which contains configurations for visual style is *layout_options*. For the initial version it will contain *max_width*, *padding*, *margin*, *background_color* and *background_image*. These are used for the overall layout and are discussed more in the context of live preview. It is likely that more attributes will be needed in the future and adding more attributes will not break backwards compatibility.

Each authentication option will have its own top-level attribute in the portal configuration. The initial version will only have the attributes *email_auth*, *SMS_auth*, and *facebook_auth*, since those are the initially supported authentication options. Each one of these contains an attribute *enabled* that indicates whether that authentication option is enabled for the portal or not. In case there are any additional settings needed for future authentication options, they can be added as attributes to that option. The *email_auth* attribute contains additional attributes for how the email address should be validated, but the validations are not implemented at the time of writing. The options are *check_valid_email*, *check_valid_mx* and *check_query_smtp_server*. They indicate whether the email should be validated at all on the server and, if so, how it should be validated.

The *content_blocks* attribute contains settings for the blocks that form the visual layout of the portal. It contains a list of blocks that are rendered from the top down in the order they appear in the list. Each block has attributes that indicate the type of block and the contents of that block. The *block_type* attribute can be *image*, *text*, or *sign_in*. The other attributes on the block are *text* and *image*; they only contain information if it is relevant for that specific block type. The *text* attribute has a *text_type* and a *content* attribute. The *text_type* attribute indicates which type of content is present in the content attribute, which is either text or HTML. The distinction between text and HTML is there because HTML can contain JavaScript and should not be injected into a webpage without first checking for malicious content, while raw text cannot be executed and is shown as it is. The *image* attribute contains image information if the block is an image block. It contains the ID of an image in the database and the visual settings for the image, such as width, height, and alignment.

The remaining attributes are metadata for the configuration. The *archived* attribute is used for archiving portals if they are no longer needed. By archiving a portal instead of just deleting it, no data is truly lost and the configuration can still be unarchived. The configuration also has attributes for *name* and *notes*. The remaining attributes, *created* and *updated*, contain the date and time of when a portal is created and the last time it was updated.

3.1.2 Portal Backend Service and Template

The backend that serves the portal is written in Golang with Go templates. It generates a server-side dynamic page for each state in the portal, based on the portal configuration. Any necessary session data is stored in memory on the server and identified with a session ID that is bundled with each request. The different states are represented with different URL paths. A URL path is the part of the URL trailing the host name in the URL as seen in Figure 3.2.

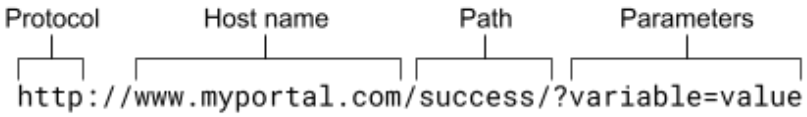

Figure 3.2: The building blocks of a uniform resource locator.

A router invokes the correct Golang function based on the HTTP method and the request URL path as seen in Figure 3.3. These functions check credentials, if relevant, and generate an HTML portal page matching the current state of the application. In the graph below, it is worth noting that the email and SMS authentication requests use POST while the Facebook authentication request uses GET. The reason for this is that email and SMS are sending form data from the portal page while the Facebook authentication request is a forward from the Facebook manual authentication flow with the necessary information as GET parameters.

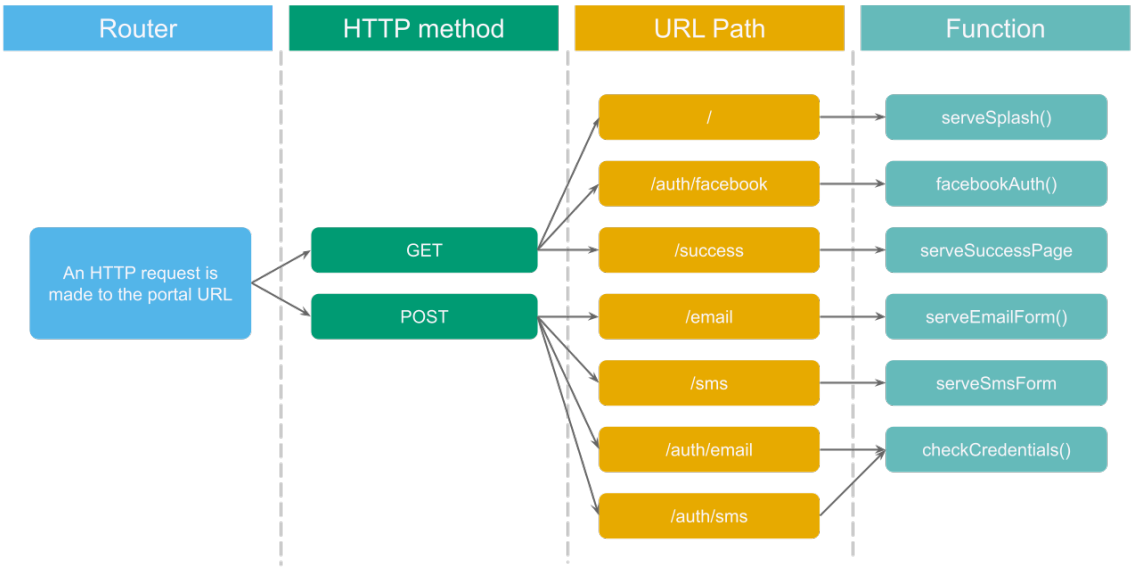

Figure 3.3: When the server receives an HTTP request, a router invokes the correct function based on HTTP method and URL path of the request.

The HTML page is generated with Go templates. Every state in the portal uses the same main template as base. The main template generates the page by looping over the list of *content_blocks* in the portal configuration. Each block is rendered according to the settings for that block. All blocks except for the sign-in block are rendered in the same manner for each state in the portal. In the default state, the sign-in block lists the sign-in options enabled in the configuration. When a user selects one of the options, the user is taken to the relevant path for that option. If email or SMS is selected, the sign-in block shows a form to enter the relevant information. If a third-party sign-in is selected, either a third-party SDK will be used to authenticate the user or the user will be redirected to that third-party for authentication. After submitting the login form or completing the third-party authentication process, the user is taken to the relevant path that validates the data and forwards the user to a success page.

3.2 Portal Editor

The portal management is incorporated into the existing Walkbase Dashboard. The portal management UI (user interface) consists of three different views. The first view is the portal listing that provides an overview of existing portals and the possibility to make new ones. The second view is the editor view that allows editing the configuration of a portal in a WYSIWYG editor. The third view, the portal assignment view, will not be implemented for the proof-of-concept version of the system. The portal assignment view will allow portals to be assigned to different customer sites and access points. At the time of writing the necessary services for assigning portals are not yet planned, however, the portal system works by manually configuring the access points as described in chapter two.

As mentioned, the portal editor will be incorporated into the existing Walkbase dashboard. Therefore, it will use the same technologies and frameworks as the dashboard. In practice, this means that the editor will be implemented as an Angular component. The portal editor will be incorporated into the dashboard under a portals tab in the Audience Intelligence page. Figure 3.4 shows a screenshot of the portal editor as it looks in the dashboard at the current stage of development.

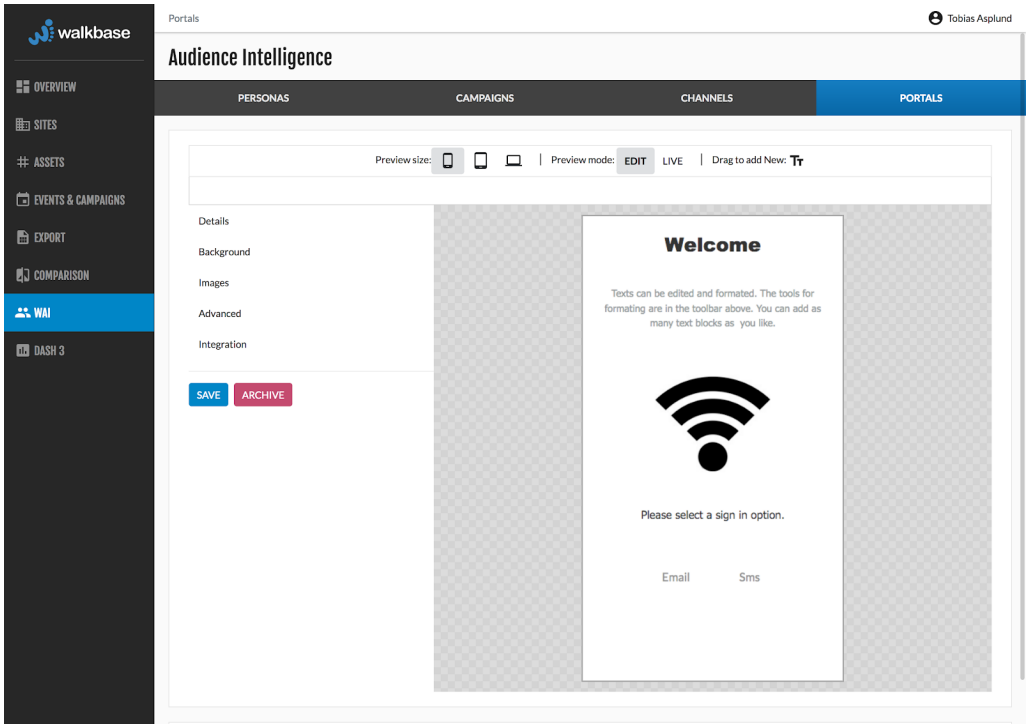

Figure 3.4: A screenshot of the full page in the Walkbase dashboard, the screenshots later on that do not include the full dashboard are cropped to only show the relevant part of the dashboard.

3.2.1 Portal Listing

The first view in the portal editor is the portal listing view. This is where the portals available to the currently signed-in user are listed. The user in this case is a user of the Walkbase analytics dashboard, the portal editor component does not need to manage user permissions since this is done behind the API used in the dashboard. In this view, the user is able to see a list of portals, open a portal to edit it, and add new portals. The portal listing can be seen in Figure 3.5.

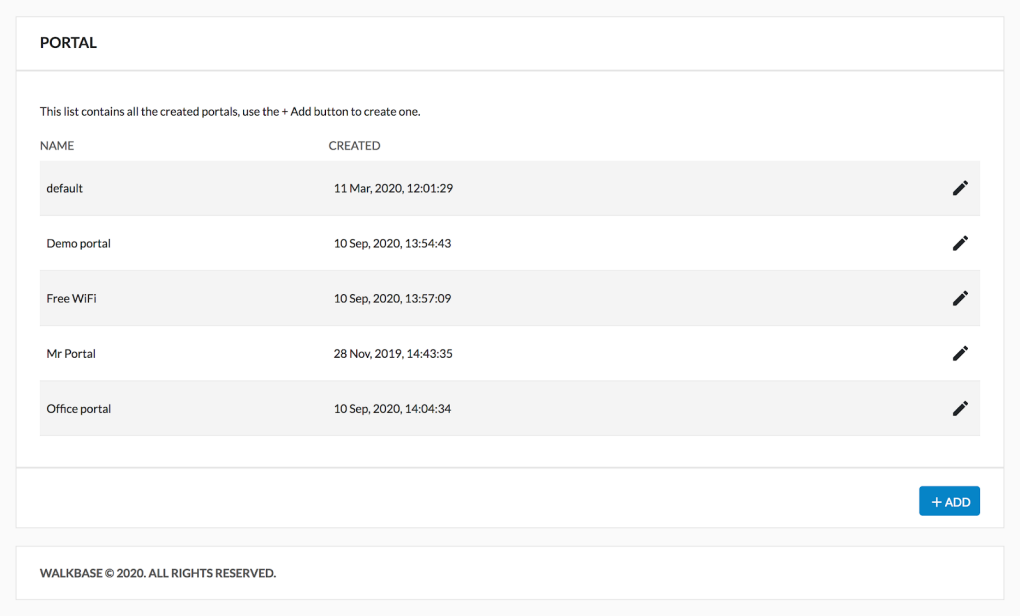

Figure 3.5: The portal listing view provides an overview of created portals and allows for the creation of new ones.

3.2.2 Settings

The settings in the editor are almost directly mapped to the configuration. The settings are separated into categories in an accordion style list. The details category contains the name and toggle boxes for the available authentication options. The background category contains a setting for the background color. Once image uploads are implemented it will support choosing a background image as well. The images category contains the available images which can be dragged and dropped onto the portal in the edit mode. The advanced settings category contains padding, margins, and maximum width. The integration category contains the URL needed to set up and access the portal. A collection of screenshots of the different settings can be seen in Figure 3.6.

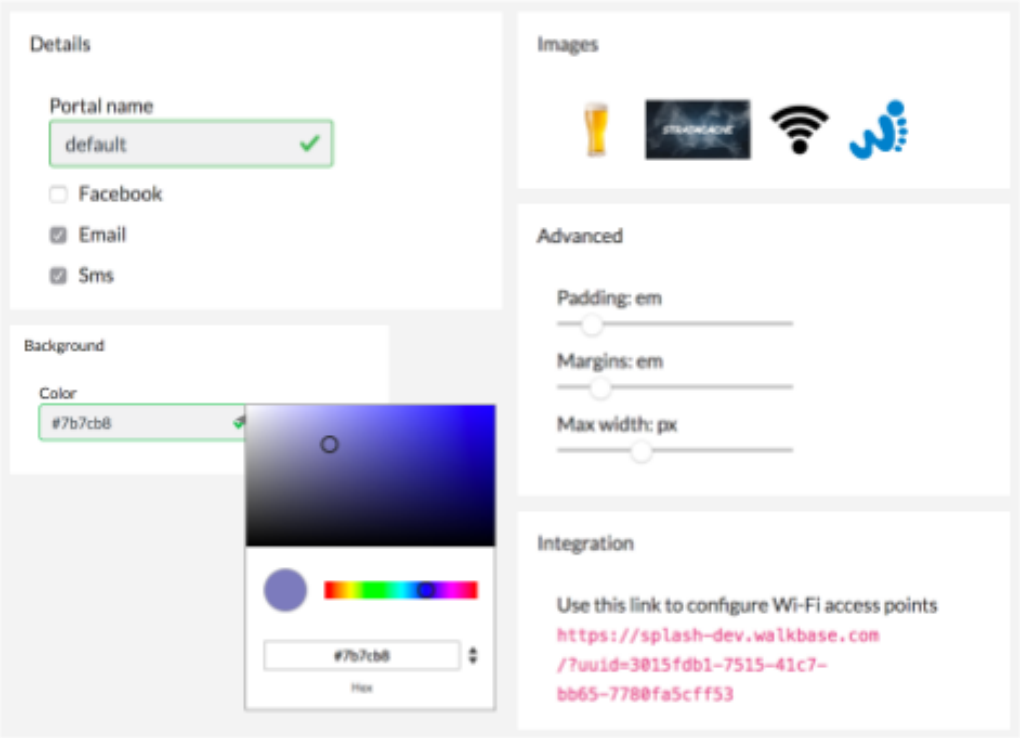

Figure 3.6: Collection of screenshots of the different settings.

3.2.3 Images

For the proof-of-concept, the image gallery only consists of a default set of statically hosted images. These images are referenced with an ID. The long-term goal is to allow for uploading and possibly editing images in the editor. The images can be dragged onto the portal in the edit mode. The specific settings for an image can be changed by clicking an image, after which the image settings appear in the top bar. The size of the image is relative to the width of the portal page. An image with a width of 100 percent would cover the whole width of the screen, up to the maximum allowed width of the portal. The alignment options specify whether the image should be left, right, or middle aligned. Figure 3.7 displays the different image settings.

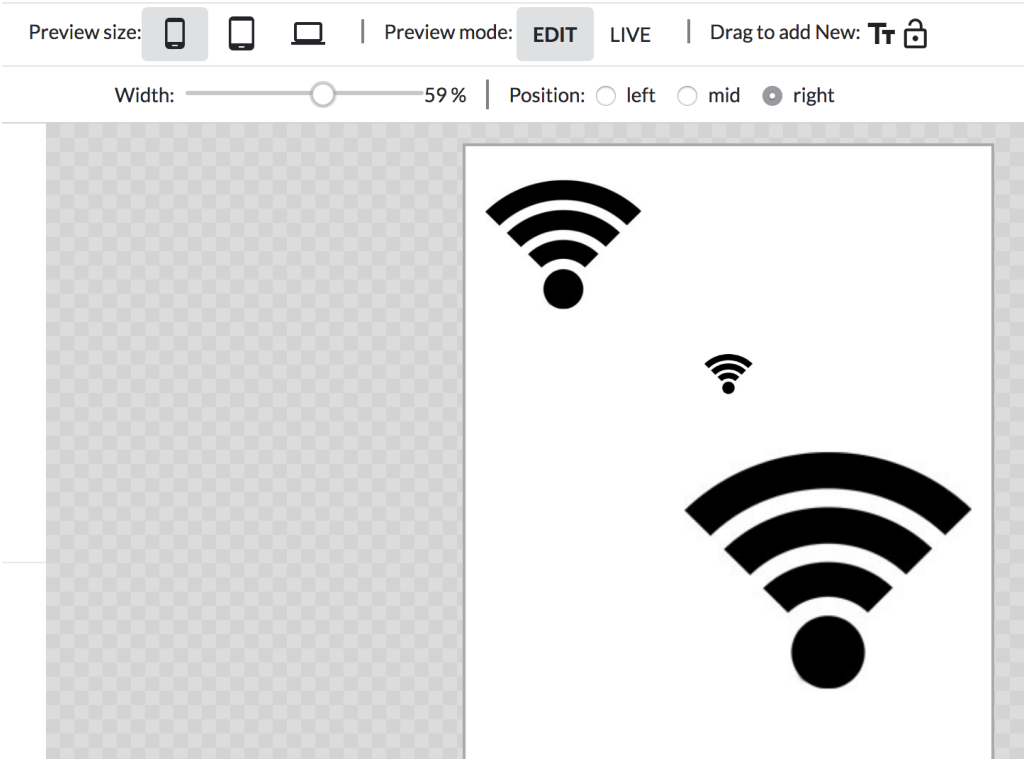

Figure 3.7: Three images added in the editor with different settings. Left, middle and right alignment in different sizes.

3.2.4 Preview

There are two preview modes in the portal editor: edit mode and live mode. The edit mode allows for editing the portal by dragging and adding blocks, with visual cues on what changes different settings make. The live preview mode loads the actual portal from the URL where the portal is served. It is loaded in an HTML iframe element as a separate page. An iframe is an HTML element that allows rendering a different webpage within a webpage. The settings are appended to the request as a GET parameter and used by the portal instead of loading a saved portal from the database. Since the portal is the actual portal running in an iframe, it allows clicking through the different steps as it would in production. What the live mode lacks in comparison to the edit mode are the visual cues for what has changed and the controls for adding and removing blocks. When settings are edited from the side panel, the page will simply reload with the new configuration. It is not

possible to drag and drop components onto the preview area when running it in live mode. Figure 3.8 and Figure 3.9 display the live and edit preview modes in the editor.

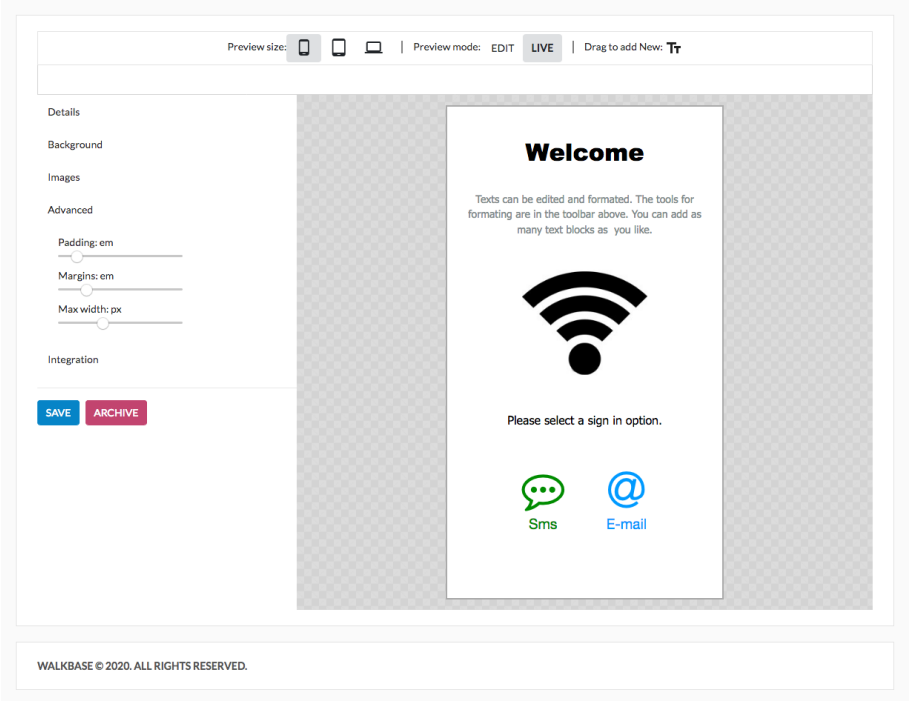

Figure 3.8: Live mode in the portal editor, a real instance of the configured portal is loaded in an iframe.

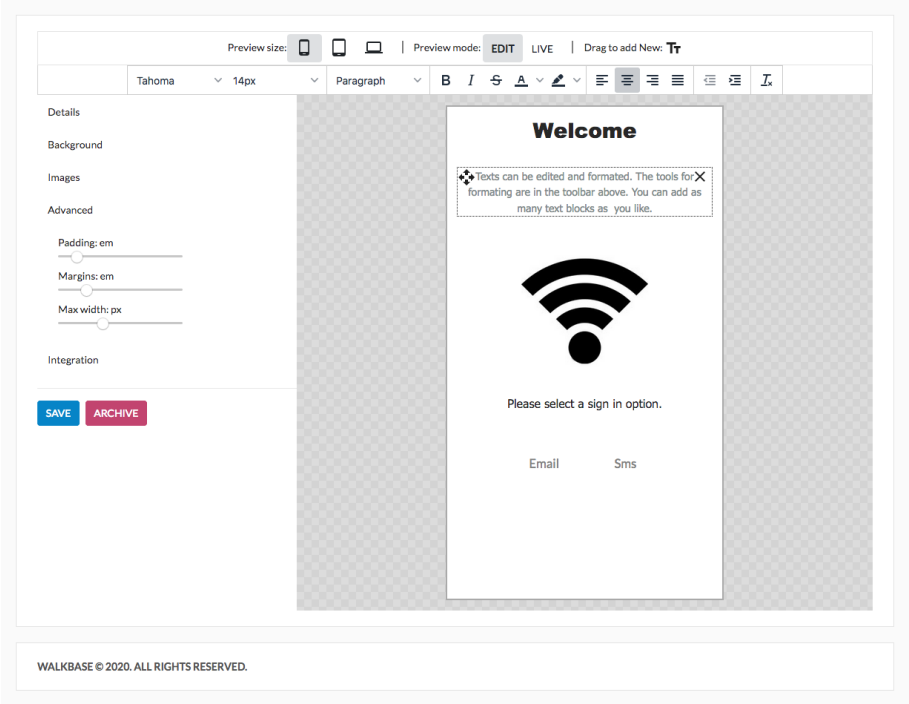

Figure 3.9: Edit mode in the portal editor, a close approximation of the real portal. The edit mode allows for editing the content directly in the preview.

To simulate different devices the editor supports scaling the preview area to simulate a phone, tablet, or desktop resolution. To be able to simulate a high resolution in the relatively small preview area, the preview is scaled to a specific factor for each size. The CSS transform *scale* [27] is used to scale the preview down while keeping the aspects it has in the intended resolution. Screenshots of the preview mode can be seen in Figure 3.10.

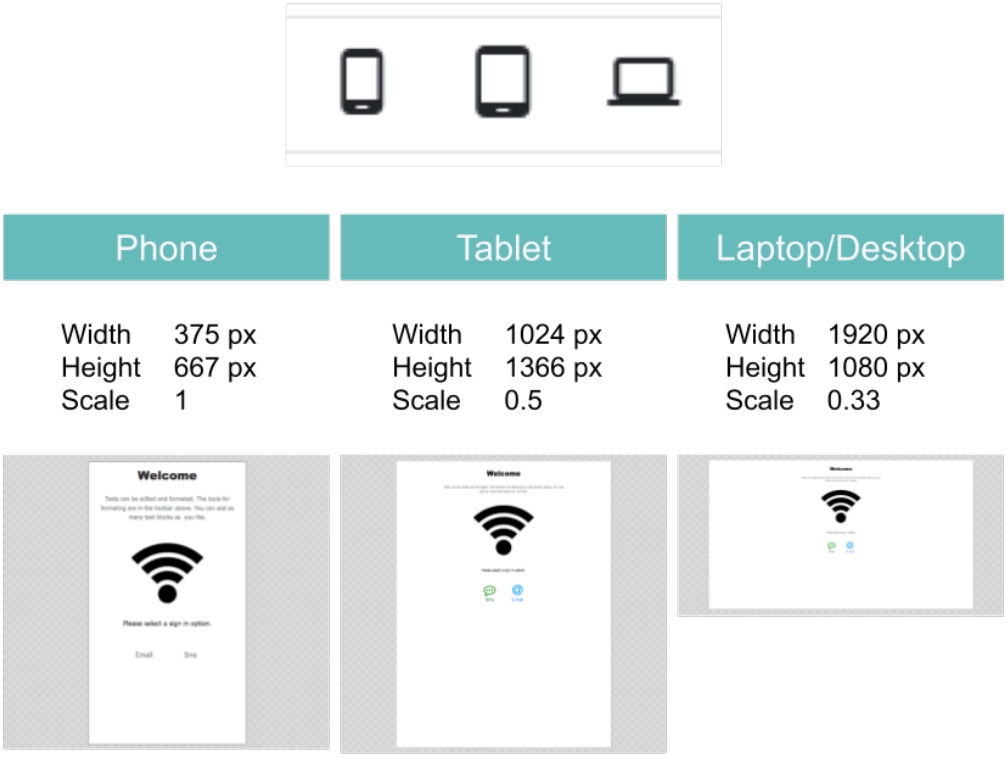

Figure 3.10: The preview buttons in the user interface with the corresponding preview sizes, scales, and screenshots.

The edit mode creates a preview of what the portal will look like, with the correct styles and content, as well as incorporated controls to add, remove, and edit items. It is not a perfect match, since it is just a representation of what the page will look like on an end-user device. The edit mode allows dragging and dropping components to add and move them. If an image block is clicked in the preview, the settings for that image are shown in the toolbar. If a text field is selected, the text editor tools are shown in the toolbar. To add a sign-in or text block while in edit mode, the user can drag one in from the top menu bar to the desired position in the preview. Image blocks can be dragged in from the image tab in the side menu. When hovering over a block, icons for removing and moving the block are displayed in the top corners, as seen in Figure 3.11. Only one sign-in block can be added to a portal.

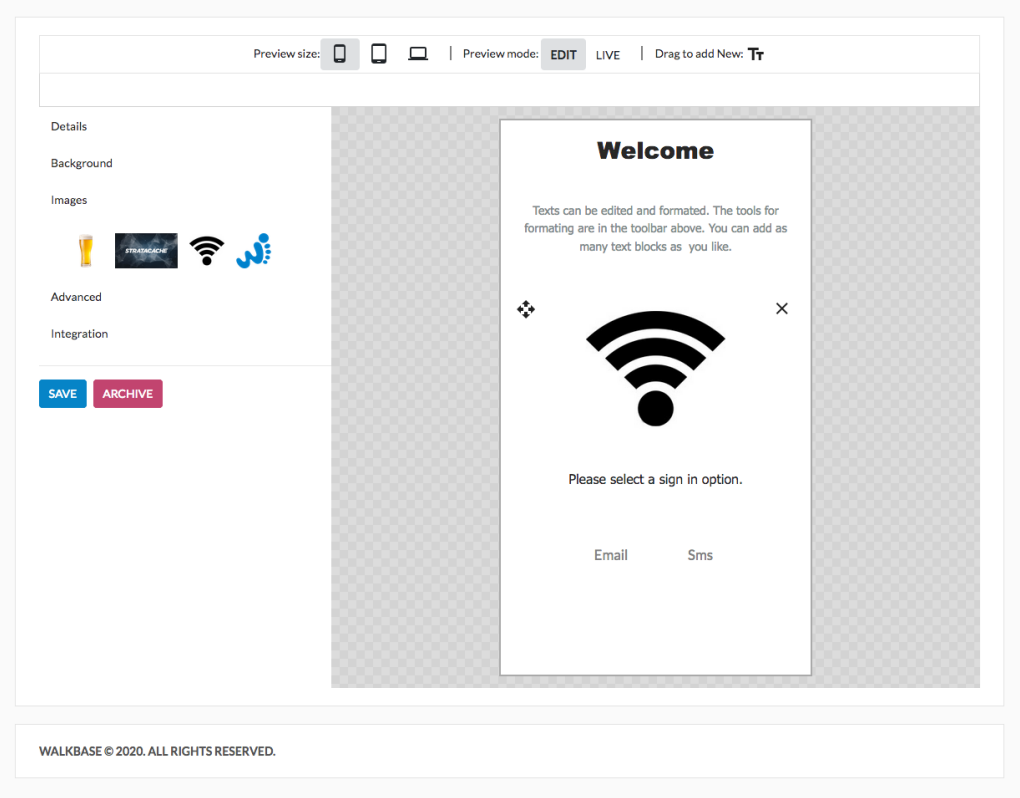

Figure 3.11: An image is hovered in the edit view, controls for moving and removing the image are visible.

Tinymce is a text editor that is used for editing the text blocks. It allows for formatting text in a way similar to common text editors. It generates HTML and styling for the edited content. When a text block is selected in the preview, text formatting options for the Tinymce editor are displayed in the top bar and the text can be edited and styled as in a normal text editor. Figure 3.12 displays a text block being edited.

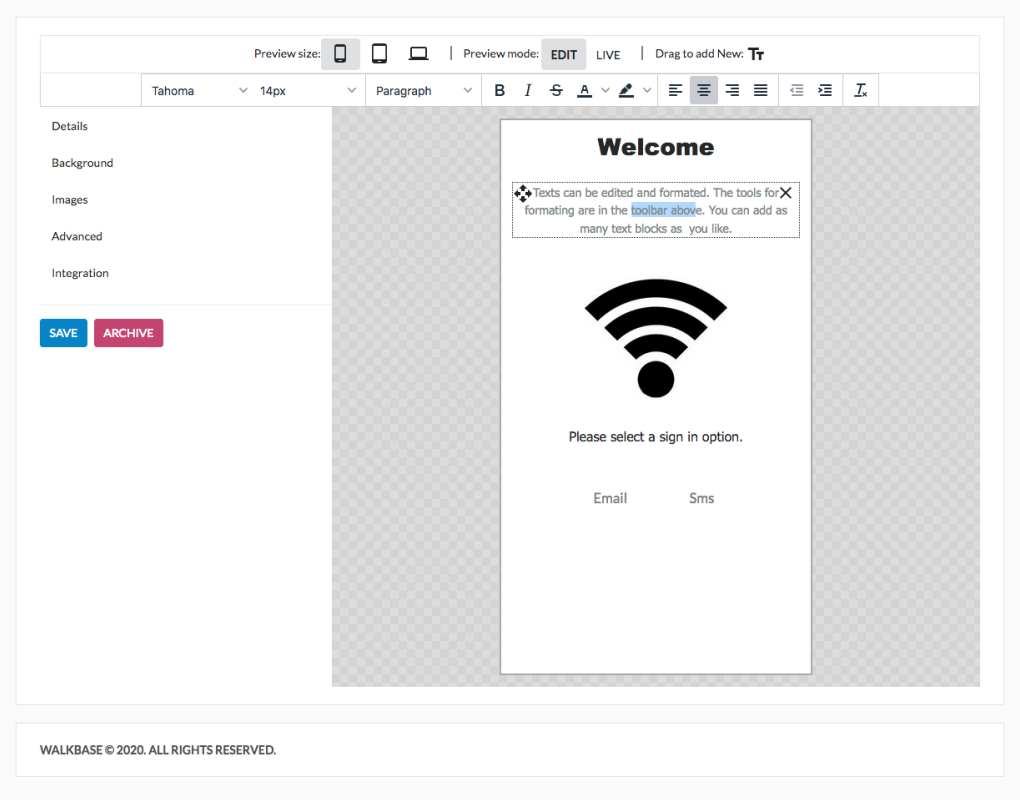

Figure 3.12: A text block is selected. The text can be edited in the portal preview and styled from the toolbar.

The edit mode supports showing visual cues when settings are changed. When the input element for paddings (Figure 3.13), margins (Figure 3.14), or maximum width (Figure 3.15, Figure 3.16) is active, the preview will highlight the area that is changing. This helps in cases where the changes might not be obvious due to other settings, such as changing padding if the width is set in such a way that the empty space on the sides is wider than the changing padding.

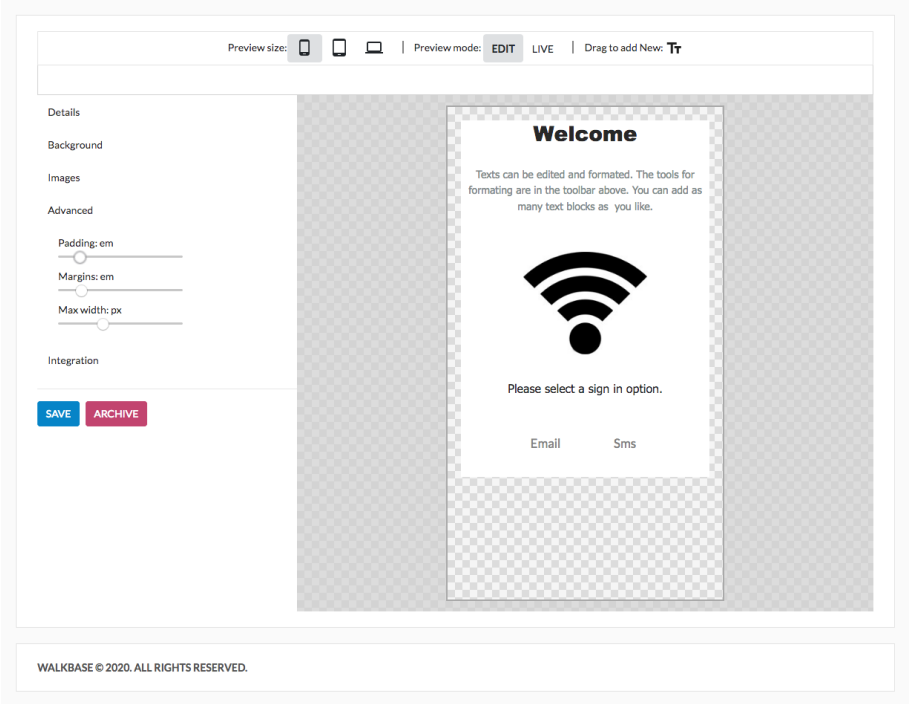

Figure 3.13: Padding is the minimum empty space between the screen edge and the content.

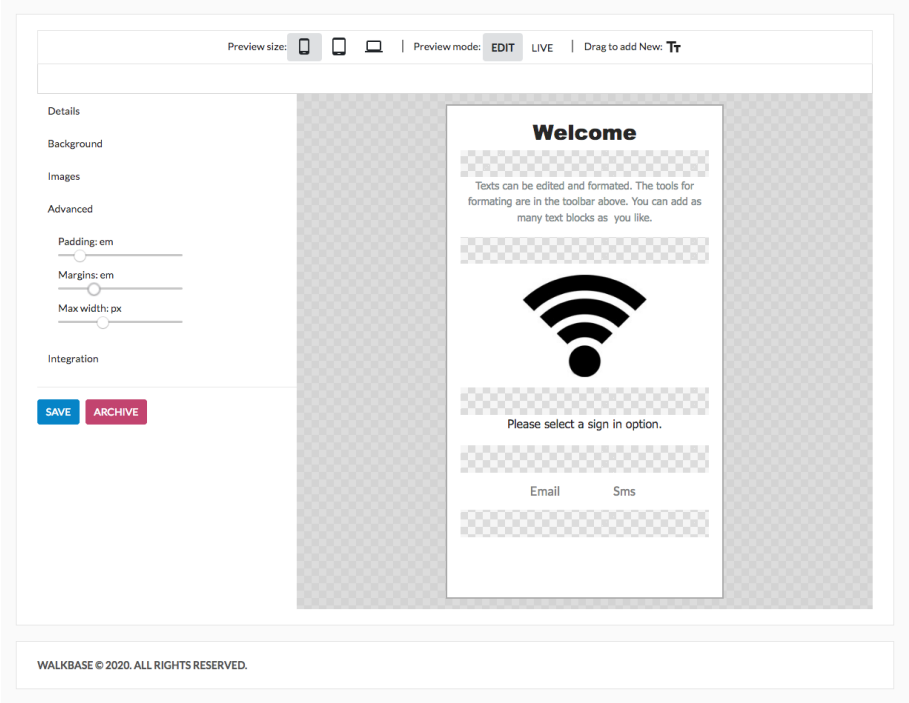

Figure 3.14: The chessboard pattern between sections visualizes the margins changing. The margins are empty space between the sections.

Figure 3.15 and Figure 3.16 show a relatively narrow and a wide width respectively. Using a narrow width will ensure that the content is laid out in a similar way on both mobile and larger devices, while a wider width will allow the content to use the available screen real estate in a more efficient manner on larger screens. A narrow width will leave the sides empty on a large screen, with the selected background. On large screens, if a very wide width is used, images might become very large and shorter texts might turn into one line instead of a few lines. It is important to use the preview modes for both smaller and larger screen sizes when using wider maximum widths.

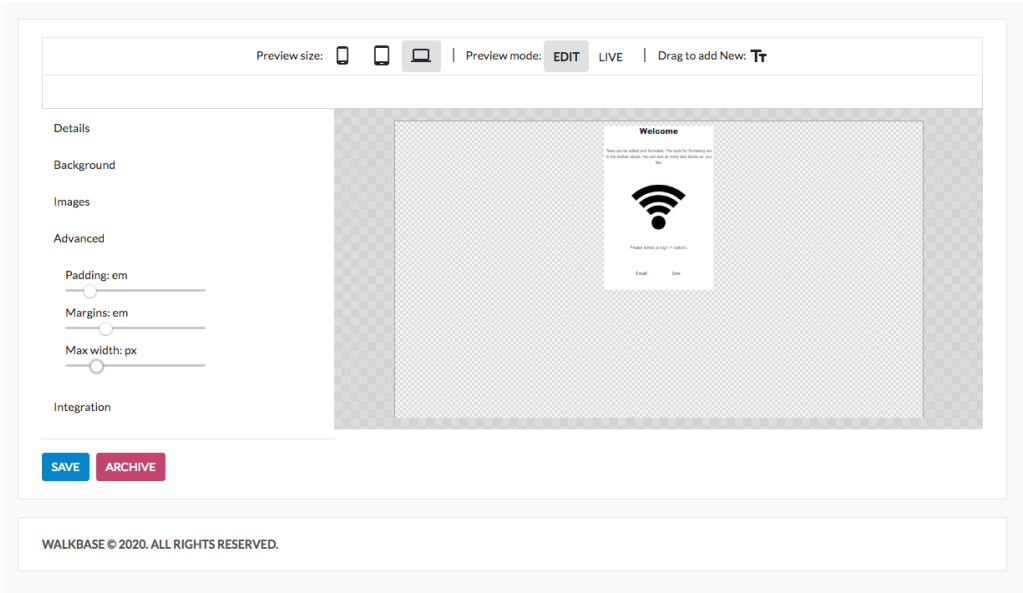

Figure 3.15: The chessboard pattern visualizes the area outside of the content. The content is restricted by the max width parameter. In this figure the max width is relatively narrow.

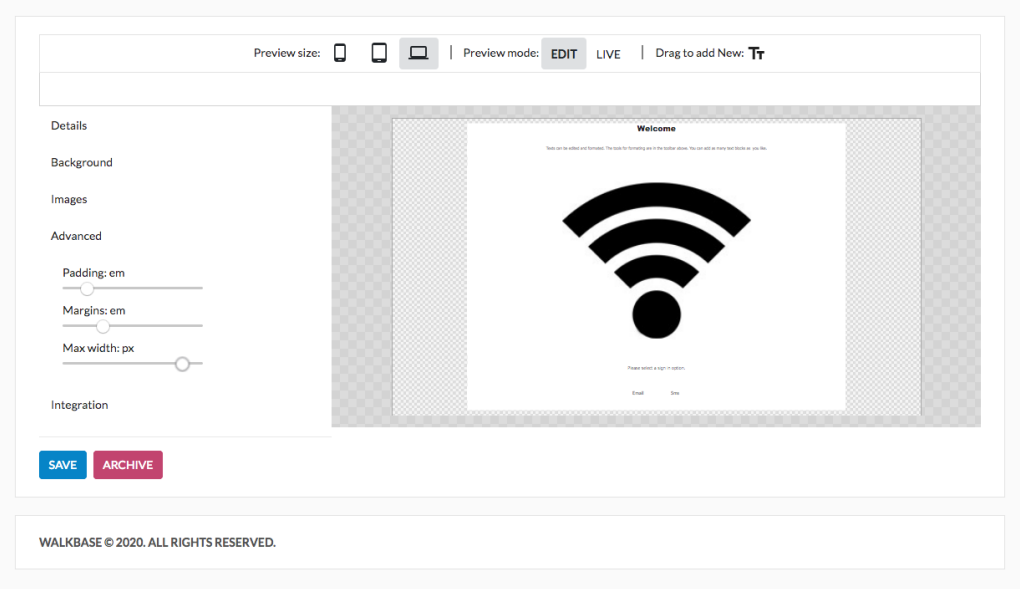

Figure 3.16: A relatively wide max width gives large images on wide screens and text sections might be displayed as a single line due to the available width.

3.2.5 APIs

To be able to save and edit the portals created in the editor there needs to be a number of new API endpoints for portal-related requests. The new endpoints for saving and editing portal content are implemented by other members of the team at Walkbase. Figure 3.17 lists the new endpoints with the corresponding HTTP method, URL, and result. The URLs are prefixed by WAI, since they are incorporated into the same APIs used for Walkbase Analytics Intelligence. The GET method requests allow for fetching all the images and portals or a specific one by ID. The POST and PUT method request takes a portal or image configuration in the form of a JSON object as request payload. If it is a POST request a new configuration is saved with a new ID, if it is a PUT request the configuration is saved with the same ID but the *revision_id* is incremented and the new revision is set as current.

METHOD	URL	RESULT
GET	/wai/wifi-portal-images	List of available images
GET	/wai/wifi-portal-images/{id}	Specific image by ID
POST	/wai/wifi-portal-images	Upload a new image
PUT	/wai/wifi-portal-images/{image__id}	Update existing portal
GET	/wai/wifi-portals	List of available portals
GET	/wai/wifi-portas/{wifi_portal__id}	Specific portal by ID
POST	/wai/wifi-portals	Create a new portal

Figure 3.17: New endpoints implemented by the Walkbase team.

4. Validation and Conclusion

The project aimed to produce a proof-of-concept version of a Wi-Fi portal system to allow Walkbase to evaluate the feasibility and scope of work for creating a fully-fledged Wi-Fi portal solution. The produced POC in its current state serves as a good base from which to develop the final product. The POC contains all the necessary building blocks except for the portal assignment functionality which is needed to be able to deploy the portal on a larger scale.

The POC covers the laid-out requirements well and also includes some features that were not included in the original POC requirements. Portals can be created, edited, and archived. The content of a portal can be customized freely with both text and predefined images. The MVP will still require backend and front-end support for uploading and managing custom images. The editor contains the requested functionality, although the layout and UX (user experience) need additional fine-tuning and optimization. Only Cisco Meraki access points are supported at the time of writing, but other common manufacturers need to be supported as well. The urgency of additional hardware support depends on what hardware is available or installed at venues where the portal system is taken into use. Additional third party authentication options need to be added, as well as completing the email and SMS authentication systems. Table 4.1 lists the remaining tasks and an estimation of the workload for each task. Beyond the POC requirements the portal editor also has modes for previewing the portal simulated for different screen sizes, which was a requirement for the MVP.

Table 4.1: Scope of work to implement a minimum viable product based on the proof-of-concept. An estimation of the percentual workload per task.

	Task	Workload	Description
1	Management tool	30%	The tool for assigning portals to access points needs to be created.
2	Image gallery	15%	APIs and UI for uploading custom images in the portal editor.
3	Editor UX	20%	The layout and user experience need to be fine-tuned.
4	Additional hardware support	15%	Support for additional access point manufacturers.
5	Third party authentication	10%	Additional third party authentication options, such as Google, Twitter, Apple.
6	Email and SMS authentication	10%	Email and SMS authentication need to be finalised.

The project was evaluated by the team at Walkbase, with a verdict available in Appendix A. The verdict concluded that the solution fulfilled the expectations of a portal system. The main evaluation consisted of setting up the system and having users connect to the Wi-Fi network though the portal. The devices used for testing were the same set of devices that are used for testing web development within the company, with both Apple and Android devices from a wide range of generations. While not as in-depth as quantitative testing, qualitative testing with users simply using the system with different devices was considered sufficient to evaluate the proof-of-concept. Based on the testing the system was deemed ready to be used for demo purposes, for example when competing for RFPs (request for proposal).

With the limited laid-out requirements the project consisted to a large extent of exploratory work. When looking back at the development process it is clear that the defined requirements could have been expanded to provide a more extensive picture of the target product. Based on the provided requirements and in-person discussions the team had a clear vision of the target solution and the vision did not change during the development process. If the project had involved outside parties

in addition to the in-house team it would have been a higher priority to establish well documented goals at the start of the project. While the project in this case was a success, having limited requirements is not a method I would recommend outside of development within well-established teams.

While the development process did not include any outside influence through surveys or user testing, this lack of outside influence is not a notable issue since the project focused on the technical solutions for the system. The produced POC can in turn be used for such surveys and customer demos to tailor the product in the future. Having a product that is modifiable and expandable provides a good base for rapid development based on feedback and future customer requirements.

To continue the work, I would suggest starting with the management tool, since it involves parties throughout the company, together with a considerable amount of planning. The APIs for the image gallery can be developed separately or in tandem with the image gallery features in the UI. The user interface of a product such as the portal editor needs to be maintained and developed continuously to cater to customer needs, hence the user interface can very well be developed further together with customers that can provide feedback based on their own requirements.

This thesis focused on the technical part of creating the Wi-Fi portal system, but to reach the full potential of the system, the data it gathers need to be incorporated into Walkbase Analytics and Walkbase Audience Intelligence. The additional data gathered from end-users will provide an even deeper insight into customer behaviour. The Wi-Fi portal page will also provide a good communication channel for reaching end-users with venue information and marketing material.

5. Summary in Swedish - Svensk sammanfattning

Design och Implementation av ett Wi-Fi-portalsystem

Introduktion

Wi-Fi-portaler utgör standarden för hur man autentiserar användare i publika nätverk. En Wi-Fi-portal är den webbsida som en användare ser när hen ansluter till ett publikt nätverk som kräver autentisering. Autentiseringen kan variera från att endast godkänna ett slutanvändaravtal till att logga in med ett socialt mediekonto eller köpa tillgång till internet. Avhandlingen redogör för processen för skapandet av en sådan portal och tillhörande infrastruktur. Målet med avhandlingen är att skapa en prototyp för Wi-Fi portalsystemet, men den behandlar även kraven för en fullständig produkt. I prototypen ingår ett verktyg för att editera portalsidor, ett system för att hantera portaler och ansluta dem till Wi-Fi accesspunkter och den nödvändiga mjukvaruinfrastrukturen som krävs för att rendera och leverera portalerna. En färdig produkt behöver stöda accesspunkter från de största tillverkarna, prototypen kommer dock endast fokusera på accesspunkter av märket Cisco Meraki.

Wi-Fi-Portal och Accesspunkt

En Wi-Fi accesspunkt är en fysisk hårdvaruenhet som möjliggör trådlös anslutning till ett nätverk. För en slutanvändare syns nätverket som ett Wi-Fi-nätverk med ett SSID (Service Set IDentifier) som syns som ett nätverksnamn. En accesspunkt skapar inte ett eget nätverk såsom en router utan utvidgar ett existerande nätverk och ger anslutna enheter tillgång till det nätverket. En accesspunkt kan konfigureras enskilt för personligt bruk eller centralt för företagsbruk. För prototypen av portalsystemet kommer accesspunkterna hanteras manuellt, men för en fullständig produkt behövs ett system för att konfigurera Wi-Fi-portaler för en hel verksamhetsplats där portaler ska användas.

Cisco Meraki-accesspunkten som används för projektet konfigureras via ett molnbaserat Meraki-användargränssnitt. I användargränssnittet finns möjligheten att konfigurera en adress varifrån Wi-Fi-portalen serveras. När en slutanvändare ansluter till ett trådlöst nätverk med en mobilenhet försöker mobilenheten kontrollera ifall nätverket ligger bakom en portal eller inte. Det gör enheten genom att göra en HTTP-förfrågan till en specifik webbadress. Ifall nätverket ligger bakom en portal kommer accesspunkten omdirigera förfrågan till portalens adress och mobilenheten har då verifierat att nätverket ligger bakom en portal. Ifall mobilenhetens förfrågan omdirigeras öppnar mobilenheten webbsidan från den slutgiltiga adressen, det vill säga portalen, i en webbläsare för att låta slutanvändaren utföra nödvändig autentisering. Ifall mobilenheten inte registrerar en omdirigering öppnas inte en webbläsare och nätverket kan användas som vanligt.

Kravspecifikation

Teknologierna som väljs för att skapa portalsystemet bör fungera väl tillsammans med redan existerande infrastruktur vid företaget. Att ta i bruk nya teknologier eller programmeringsspråk kräver att de är märkbart bättre än de teknologier som redan används inom företaget. Lösningen bör fungera väl även på äldre mobilenheter för att maximera den nåbara användarbasen. Det är dock inte ett krav att stöda enheter som har en mycket liten användarbas ifall enheterna i fråga kräver lösningar som gör systemet mer komplext.

Designprocess

En handfull teknologier har valts ut baserad på popularitet inom branchen och kännedom inom företaget. Teknologierna kan grupperas som sådana som renderar portalen på servern och sådana som renderar portalen i webbläsaren på mobilenheten. Hur portalerna sparas och hanteras är beroende av vilka teknologier som används för att skapa portalsystemet. Portalediteraren ska bli det verktyg som används för att skapa och editera portaler. Portalediteraren kommer att integr-

eras i företagets existerande användargränssnitt och kommer därför vara baserat på samma teknologier som använts där. Den kommer byggas som en Angular-komponent och ska även kunna lista existerande portaler och arkivera gamla eller oanvända portaler. Av teknologierna som utvärderats för själva portalen valdes programmeringsspråket Golang med sina text- och HTML-mallpaket. Denna lösning möjliggör rendering på servern och minimerar därför möjliga problem som kan uppkomma vid rendering på en mobilenhet. Själva portalkonfigurationen blir ett JSON-objekt som innehåller all information som behövs för att rendera portalen. Portalkonfigurationen skickas från portalediteraren till en databas, varifrån den begärs av Golang-applikationen som renderar och serverar portalen till slutanvändaren.

Slutsats

Resultatet av avhandlingen fyller kraven för en prototyp bra och även en del av kraven för en fullständig produkt. En hel del fortsatt arbete krävs dock för att nå slutmålet, i nuläget saknas hela den automatiserade hanteringen av accesspunkter, vilket kan kräva en betydande del arbete att implementera för att stöda flera tillverkares hårdvara. Användargränssnittet i portalediteraren har alla de listade funktionerna för prototypen, men den grafiska delen och användarvänligheten behöver utvecklas för en slutgiltig produkt.

References

[1] GSMA, "The mobile economy 2019," GSMA, 2019.

[2] Niall McCarthy, "The cost of mobile internet around the world," *Forbes*, 2019. [Online]. Available: https://www.forbes.com/sites/niallmccarthy/2019/03/05/the-cost-of-mobile-internet-around-the-world-infographic/ (visited on 05/18/2021).

[3] J. L. Lloyd Gofton, "Purple wifi – using wifi in public places," 2014. [Online]. Available: https://www.realwire.com/writeitfiles/Using-WiFi-in-public-Places-Survey-data-overview-Purple-WiFi.pdf (visited on 05/18/2021).

[4] Stratacache Oy. (2021). "Audience intelligence," [Online]. Available: https://www.walkbase.com/audience-intelligence/ (visited on 05/18/2021).

[5] Apple Inc. (2018). "Use captive wi-fi networks on your iphone, ipad, or ipod touch," [Online]. Available: https://support.apple.com/en-us/HT204497 (visited on 05/18/2021).

[6] Google LLC. (2021). "Captive portal api support," [Online]. Available: https://developer.android.com/about/versions/11/features/captive-portal (visited on 05/18/2021).

[7] L. Miarka. (2020). "Dns fix to keep android splash page and the captive portal notification active," [Online]. Available: https://socifi-doc.atlassian.net/wiki/spaces/SC/pages/94371841/ (visited on 05/18/2021).

[8] Mozilla and individual contributors. (2021). "Http," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP (visited on 05/18/2021).

[9] E. K. W. Kumari. (2019). "Captive-portal identification in dhcp / ra draft-ekwk-capport-rfc7710bis-02," [Online]. Available: https://tools.ietf.org/html/draft-ekwk-capport-rfc7710bis-02 (visited on 05/18/2021).

[10] Cisco Systems, Inc. (2020). "MR - Wireless LAN," [Online]. Available: https://documentation.meraki.com/MR (visited on 05/18/2021).

[11] ——, (2020). “Configuring a custom-hosted splash page to work with the meraki cloud,” [Online]. Available: https://documentation.meraki.com/General_Administration/Cross-Platform_Content/Configuring_a_Custom-Hosted_Splash_Page_to_Work_with_the_Meraki_Cloud (visited on 05/18/2021).

[12] ——, (2020). “Walled garden,” [Online]. Available: https://documentation.meraki.com/zGeneral_Administration/Cross-Platform_Content/Walled_Garden (visited on 05/18/2021).

[13] WHATWG. (2021). “Html standard,” [Online]. Available: https://html.spec.whatwg.org/ (visited on 05/18/2021).

[14] G. Inc. (2021). “Https://angular.io/,” [Online]. Available: https://angular.io/ (visited on 05/18/2021).

[15] StatCounter. (2021). “Statcounter global stats - browser, os, search engine including mobile usage share,” [Online]. Available: https://gs.statcounter.com/ (visited on 05/18/2021).

[16] Y. Xing, J. Huang, and Y. Lai, “Research and analysis of the front-end frameworks and libraries in e-business development,” in *Proceedings of the 2019 11th International Conference on Computer and Automation Engineering*, ser. ICCAE 2019, Perth, WN, Australia: Association for Computing Machinery, 2019, 68–72, ISBN: 9781450362870. [Online]. Available: https://doi.org/10.1145/3313991.3314021.

[17] Google Inc. (2021). “Angular - browser support,” [Online]. Available: https://angular.io/guide/browser-support (visited on 05/18/2021).

[18] JavaScript.com. (2021). “Javascript.com,” [Online]. Available: https://www.javascript.com/ (visited on 05/18/2021).

[19] OpenJS Foundation. (2021). “Jquery,” [Online]. Available: https://jquery.com/ (visited on 05/18/2021).

[20] Hugo Authors. (2021). “Hugo documentation,” [Online]. Available: https://gohugo.io/documentation/ (visited on 05/18/2021).

[21] Golang.org. (2021). “The go programming language,” [Online]. Available: https://golang.org/ (visited on 05/18/2021).

[22] Django Sofrware Foundation. (2021). “The web framework for perfectionists with deadlines | django,” [Online]. Available: https://www.djangoproject.com/ (visited on 05/18/2021).

[23] Golang.org. (2021). “Package template,” [Online]. Available: https://golang.org/pkg/text/template/ (visited on 05/18/2021).

[24] ——, (2021). “Package template,” [Online]. Available: https://golang.org/pkg/html/template/ (visited on 05/18/2021).

[25] Facebook. (2021). “Manually build a login flow - facebook login,” [Online]. Available: https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow/v2.3 (visited on 05/18/2021).

[26] Facebook. (2021). “Facebook for developers,” [Online]. Available: https://developers.facebook.com/ (visited on 05/18/2021).

[27] Mozilla and individual contributors. (2021). “Scale(),” [Online]. Available: "https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function/scale()" (visited on 05/18/2021).

A. Wi-Fi Portal Evaluation

Wi-Fi Portal Evaluation

The system that Tobias built was part of a bigger project where we are building a suite of solutions for customer engagement. Wi-Fi portals being one of the key pieces.

The challenges of building a portal system that can be used on multiple Wi-Fi systems are many. Each mobile device and web browser have different ways of handling the portal sign-in process, which makes creating a cohesive experience difficult. In addition to this, each Wi-Fi system has a different method for handling Wi-Fi sign-on.

The solution that was delivered as part of the work done for this thesis more than fulfilled our expectations of a portal service with user customizable landing pages and different authentication methods. We were able to run the solution in our demo environment and demo it to internal stakeholders and potential customers. The project is built so that it can be expanded to additional Wi-Fi systems if the need arises. For Stratacache the main outcome from this project is that we now have a system that we can use to compete for RFP's where this is a requirement.

Björn Sjölund

VP Product & Data

Stratacache Oy